МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждения высшего образования

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**ЛАБОРАТОРНАЯ РАБОТА № 1**

по дисциплине

**«Объектно-ориентированное программирование»**

на тему:

**«Классы и объекты С++»**

*Вариант № 2*

Выполнил:

Студент группы

КТбо2-7

Шубенков С. С.

Проверил:

Тарасов С. А.

Оценка

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Таганрог 2020

# **1 ФОРМУЛИРОВКА ЗАДАНИЯ**

Согласно варианту задания, требуется определить класс, создающий объект типа обыкновенная дробь (Fraction) с методами сравнения объектов, изменения, сложения, умножения, деления, сокращения.

При выполнении работы необходимо: − разработать соответствующие классы, конструкторы, поля и методы; − поля класса сделать закрытыми; для чтения и изменения их значений определить открытые методы; – предусмотреть во всех вариантах консольный ввод данных для создания объектов и консольный вывод результатов. – во всех вариантах необходимо использовать хотя бы один раз блоки try catch. Это можно сделать для контроля арифметических ошибок, для проверки существования файлов и т. п. Отсутствие такового карается дополнительным уменьшением оценки. – во всех перечисленных вариантах запрещено пользоваться контейнерами STL, кроме string.

# **2 СПЕЦИФИКАЦИЯ КЛАССОВ**

class Fraction {

public:

int GetNumerator() const;

int GetDenumerator() const;

void SetFraction(const int& num, const int& den = 1);

void AddFraction(const Fraction& fr);

void MultiplyFraction(const Fraction& fr);

void SubtractFraction(Fraction fr);

void DivisionFraction(Fraction fr);

void ReductionFraction();

Fraction operator+ (const Fraction& fr);

Fraction operator\* (const Fraction& fr);

Fraction operator- (const Fraction& fr);

Fraction operator/ (const Fraction& fr);

bool operator> (const Fraction& fr);

bool operator<= (const Fraction& fr);

bool operator< (const Fraction& fr);

bool operator>= (const Fraction& fr);

bool operator== (const Fraction& fr);

bool operator!= (const Fraction& fr);

Fraction() = default;

Fraction(const int& num, const int& den = 1);

private:

int \_numerator;

int \_denumerator ;

void CorrectedMinuses();

inline void CheckDenominator();

};

class Console

{

public:

void Run();

~Console();

private:

Fraction\* \_result;

int \_numberFr;

int \_numberFrNow;

int \_menuItemNumber;

Fraction \_first;

Fraction \_second;

void SetNumFr();

void Menu();

void PrintFr(const Fraction& fr);

void ChangeFr(Fraction& fr);

void Input(int& Int);

void SaveFr(const Fraction& fr);

bool PrintArrayFr();

void Actions();

void MenuActions();

void LocationSaveFr(const Fraction& fr);

void ComparisonFr();

Fraction& SelectFrArray();

};

# **4 ДИАГРАММА КЛАССОВ**

UML диаграмма классов изображена на рисунке 1.

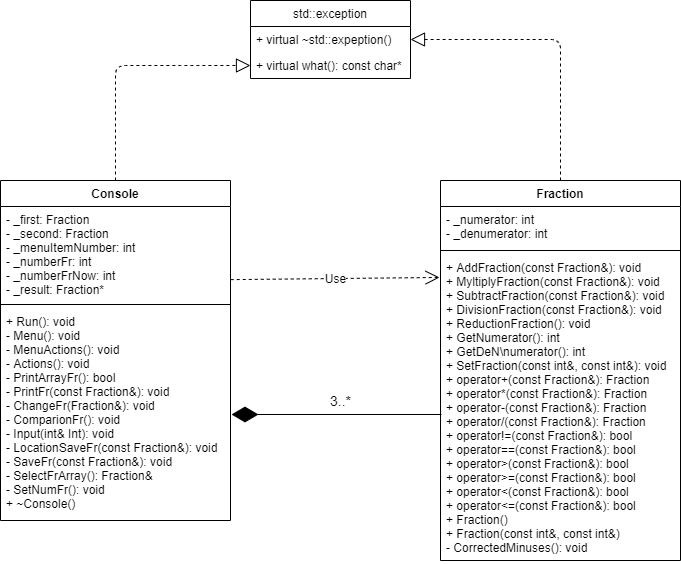


Рисунок 1 – Диаграмма классов

**ЛИСТИНГ ПРОГРАММЫ**

Frachion.h

#pragma once

#include <algorithm>

class Fraction {

public:

int GetNumerator() const { return \_numerator; } //получение числителя

int GetDenumerator() const { return \_denumerator; } //получение знаменателя

void SetFraction(const int& num, const int& den = 1) //изменение дроби

{

if (den == 0)

{

throw std::exception("The denominator is zero");

}

\_numerator = num;

\_denumerator = den;

CorrectedMinuses();

}

void AddFraction(const Fraction& fr); //прибавление дроби

void MultiplyFraction(const Fraction& fr); //умножение дроби

void SubtractFraction(Fraction fr); //вычитание дроби

void DivisionFraction(Fraction fr); //деление дробей

void ReductionFraction(); //сокращение дроби

Fraction operator+ (const Fraction& fr);

Fraction operator\* (const Fraction& fr);

Fraction operator- (const Fraction& fr);

Fraction operator/ (const Fraction& fr);

bool operator> (const Fraction& fr);

bool operator<= (const Fraction& fr);

bool operator< (const Fraction& fr);

bool operator>= (const Fraction& fr);

bool operator== (const Fraction& fr);

bool operator!= (const Fraction& fr);

Fraction() = default;

Fraction(const int& num, const int& den = 1)

{

if (den == 0)

{

throw std::exception("The denominator is zero");

}

\_numerator = num;

\_denumerator = den;

CorrectedMinuses();

}

private:

int \_numerator = 0;

int \_denumerator = 1;

//функция приведения отрицальных дробей к "стандартному" виду

//(минус всегда только в числителе)

void CorrectedMinuses();

};

Fraction.cpp

#include "Fraction.h"

void Fraction::AddFraction(const Fraction& fr)

{

if (!fr.\_numerator) return;

\_numerator = \_numerator \* fr.\_denominator + \_denominator \* fr.\_numerator;

\_denominator = \_denominator \* fr.\_denominator;

ReductionFraction();

}

void Fraction::MultiplyFraction(const Fraction& fr)

{

if (!fr.\_numerator)

{

\_numerator = 0;

\_denominator = 1;

return;

}

\_numerator = \_numerator \* fr.\_numerator;

\_denominator = \_denominator \* fr.\_denominator;

ReductionFraction();

}

void Fraction::SubtractFraction(Fraction fr)

{

fr.\_numerator \*= -1;

AddFraction(fr);

}

void Fraction::DivisionFraction(Fraction fr)

{

if (!fr.\_numerator)

{

throw std::exception("Division by zero");

}

std::swap(fr.\_numerator, fr.\_denominator);

MultiplyFraction(fr);

}

void Fraction::ReductionFraction()

{

if (\_numerator == 0)

{

\_denominator = 1;

return;

}

//находим общий делитель

int a = \_numerator;

int b = \_denominator;

while (b) {

a %= b;

std::swap(a, b);

}

a = abs(a);

\_numerator /= a;

\_denominator /= a;

}

Fraction Fraction::operator+ (const Fraction& fr)

{

Fraction res = \*this;

res.AddFraction(fr);

return res;

}

Fraction Fraction::operator\* (const Fraction& fr)

{

Fraction res = \*this;

res.MultiplyFraction(fr);

return res;

}

Fraction Fraction::operator- (const Fraction& fr)

{

Fraction res = \*this;

res.SubtractFraction(fr);

return res;

}

Fraction Fraction::operator/ (const Fraction& fr)

{

Fraction res = \*this;

res.DivisionFraction(fr);

return res;

}

bool Fraction::operator> (const Fraction& fr)

{

if (\_numerator \* fr.\_denominator > \_denominator \* fr.\_numerator)

return true;

return false;

}

bool Fraction::operator<= (const Fraction& fr)

{

return !(\*this > fr);

}

bool Fraction::operator< (const Fraction& fr)

{

if (\_numerator \* fr.\_denominator < \_denominator \* fr.\_numerator)

return true;

return false;

}

bool Fraction::operator>= (const Fraction& fr)

{

return !(\*this < fr);

}

bool Fraction::operator== (const Fraction& fr)

{

if (\_numerator \* fr.\_denominator == \_denominator \* fr.\_numerator)

return true;

return false;

}

bool Fraction::operator!= (const Fraction& fr)

{

return !(\*this == fr);

}

void Fraction::CorrectedMinuses()

{

if (\_denominator < 0)

{

\_numerator = -\_numerator;

\_denominator = -\_denominator;

}

}

Console.h

#pragma once

#include <iostream>

#include "Fraction.h"

using namespace std;

class Console

{

public:

void Run(); //запуск консоли

~Console()

{

delete[] \_results;

}

private:

Fraction\* \_results = nullptr; //массив дробей

int \_numberFr = 0; //размер массива

int \_numberFrNow = 0; //количество запис. в мас. дробей

int \_menuItemNumber = -1; //номер пункта меню

Fraction \_first; //первая дробь

Fraction \_second; //вторая дробь

void SetNumFr(); //установка размера массива

void Menu(); //вывод главного меню

void PrintFr(const Fraction& fr); //вывод дроби

void ChangeFr(Fraction& fr); //изменение дроби

void Input(int& Int); //ввод

void SaveFr(const Fraction& fr); //сохранение дроби

bool PrintArrayFr(); //вывод массива дробей

void Actions(); //меню действий с дробями

void MenuActions(); //вывод меню

void LocationSaveFr(const Fraction& fr); //выбор места сох. др.

void ComparisonFr(); //сравнение дробей

Fraction& SelectFrArray(); //выбор дроби из массива

};

Console.cpp

#include "Console.h"

void Console::Run()

{

SetNumFr();

while (\_menuItemNumber)

{

Menu();

Input(\_menuItemNumber);

switch (\_menuItemNumber)

{

case 1:

ChangeFr(\_first);

break;

case 2:

ChangeFr(\_second);

break;

case 3:

Actions();

break;

case 4:

SaveFr(\_first);

break;

case 5:

SaveFr(\_second);

break;

case 6:

(void)PrintArrayFr();

break;

case 7:

SetNumFr();

break;

case 0:

break;

default:

system("cls");

cout << "Incorrect menu item number. Try again\n\n";

}

}

}

void Console::SetNumFr()

{

delete[] \_results;

\_numberFr = 0;

system("cls");

cout << "Enter the array size to save fractions\n";

while (\_numberFr == 0)

{

cout << '>';

Input(\_numberFr);

if (\_numberFr != 0)

{

try

{

\_results = new Fraction[\_numberFr];

\_numberFrNow = 0;

system("cls");

cout << "An array of " << \_numberFr << " is created\n\n";

}

catch (bad\_array\_new\_length)

{

if (\_numberFr < 0)

{

cout << "Error. Enter a number greater than 0\n";

}

else

{

cout << "Error. Array size is too big. Please enter a smaller size\n";

}

\_numberFr = 0;

}

}

else

{

cout << "Enter a size greater than 0\n";

}

}

}

inline void Console::Menu()

{

cout << "1. Enter the first fraction\t= ";

PrintFr(\_first);

cout << "\n2. Enter the second fraction\t= ";

PrintFr(\_second);

cout << "\n3. Actions with a fraction\n";

cout << "4. Save the first fraction\n";

cout << "5. Save the second fraction\n";

cout << "6. View saved fractions\n";

cout << "7. Re-create an array of fractions\n";

cout << "0. Complete the program\n>";

}

void Console::PrintFr(const Fraction& fr)

{

cout << fr.GetNumerator() << '/' << fr.GetDenumerator();

}

void Console::ChangeFr(Fraction& fr)

{

int num = 0, den = 1;

system("cls");

cout << "1. Select a fraction from the array\n";

cout << "2. Introduce a new fraction\n";

\_menuItemNumber = -1;

while (\_menuItemNumber != 1 && \_menuItemNumber != 2)

{

cout << '>';

Input(\_menuItemNumber);

switch (\_menuItemNumber)

{

case 1:

if (PrintArrayFr())

{

fr = SelectFrArray();

system("cls");

}

break;

case 2:

cout << "Enter numerator\n>";

Input(num);

cout << "Enter denumerator\n>";

Input(den);

system("cls");

try

{

fr.SetFraction(num, den);

fr.ReductionFraction();

system("cls");

}

catch (exception er)

{

cout << "Error. " << er.what() << ". Try again\n\n";

cout << "1. Select a fraction from the array\n";

cout << "2. Introduce a new fraction\n";

\_menuItemNumber = 0;

}

break;

default:

cout << "Incorrect menu item number. Try again\n";

break;

}

}

}

void Console::Input(int& Int)

{

cin >> Int;

if (cin.fail())

{

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

}

void Console::SaveFr(const Fraction& fr)

{

system("cls");

if (\_numberFrNow == \_numberFr)

{

\_results[\_numberFr - 1] = fr;

cout << "Attention. The array is full. The answer will rewrite the last answer. Re-create the array\n\n";

return;

}

\_results[\_numberFrNow] = fr;

\_numberFrNow++;

}

bool Console::PrintArrayFr()

{

system("cls");

if (\_numberFrNow == 0)

{

cout << "The array is empty\n\n";

return false;

}

for (int i = 0; i < \_numberFrNow; i++)

{

cout << i + 1 << ") ";

PrintFr(\_results[i]);

cout << '\n';

}

cout << '\n';

return true;

}

inline void Console::Actions()

{

system("cls");

\_menuItemNumber = 0;

Fraction res;

while (!\_menuItemNumber)

{

MenuActions();

Input(\_menuItemNumber);

switch (\_menuItemNumber)

{

case 1:

res = \_first + \_second;

break;

case 2:

res = \_first - \_second;

break;

case 3:

res = \_first \* \_second;

break;

case 4:

try

{

res = \_first / \_second;

}

catch (exception er)

{

cout << "Error. " << er.what() << ".\n";

}

break;

case 5:

ComparisonFr();

break;

default:

system("cls");

cout << "Incorrect menu item number. Try again\n\n";

\_menuItemNumber = 0;

break;

}

}

if (\_menuItemNumber < 5)

{

LocationSaveFr(res);

system("cls");

}

}

inline void Console::MenuActions()

{

cout << "1. Fold the fraction\n";

cout << "2. Subtract fractions\n";

cout << "3. Multiply fractions\n";

cout << "4. Split fractions\n";

cout << "5. Compare fractions\n>";

}

void Console::LocationSaveFr(const Fraction& fr)

{

\_menuItemNumber = 0;

cout << "\nResult = ";

PrintFr(fr);

cout << "\n\n1. Save the result in the first fraction\n";

cout << "2. Save the result to an array\n>";

while (\_menuItemNumber != 1 && \_menuItemNumber != 2)

{

Input(\_menuItemNumber);

switch (\_menuItemNumber)

{

case 1:

\_first = fr;

break;

case 2:

SaveFr(fr);

break;

default:

cout << "Incorrect menu item number. Try again\n>";

break;

}

}

}

inline void Console::ComparisonFr()

{

bool comparison = true;

system("cls");

\_menuItemNumber = 0;

cout << "1. Compare the first shot with the second\n";

cout << "2. Compare the first fraction with an array of fractions\n";

cout << "3. Compare the second fraction with an array of fractions\n>";

Fraction fr1, fr2;

while (\_menuItemNumber < 1 || \_menuItemNumber > 3)

{

Input(\_menuItemNumber);

switch (\_menuItemNumber)

{

case 1:

fr1 = \_first;

fr2 = \_second;

break;

case 2:

if (PrintArrayFr())

{

fr1 = \_first;

fr2 = SelectFrArray();

}

else

{

comparison = false;

}

break;

case 3:

if (PrintArrayFr())

{

fr1 = \_second;

fr2 = SelectFrArray();

}

else

{

comparison = false;

}

break;

default:

cout << "Incorrect number. Try again\n>";

break;

}

}

if (comparison)

{

cout << '\n';

PrintFr(fr1);

if (fr1 == fr2)

{

cout << " = ";

}

else

if (fr1 > fr2) cout << " > ";

else cout << " < ";

PrintFr(fr2);

cout << "\n\n5. Exit to the main menu\n";

cout << "0. Exit the action menu\n>";

\_menuItemNumber = -1;

while (\_menuItemNumber != 5 && \_menuItemNumber != 0)

{

Input(\_menuItemNumber);

switch (\_menuItemNumber)

{

case 5:

case 0:

system("cls");

break;

default:

cout << "Incorrect menu item number. Try again\n>";

break;

}

}

}

else

{

\_menuItemNumber = 5;

}

}

Fraction& Console::SelectFrArray()

{

int number = 0;

cout << "Select a fraction\n>";

while (!number)

{

Input(number);

if (number < 0 && number > \_numberFrNow)

{

number = 0;

cout << "Incorrect number. Try again\n>";

}

}

return \_results[number - 1];

}

main.cpp

#include <vld.h>

#include "Fraction.h"

#include "Console.h"

int main()

{

Console con;

con.Run();

return 0;

}